VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by
Pradhan Sagar K
1BM23CS237

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by **Pradhan Sagar K (1BM23CS237)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Lakshmi Neelima M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab - 1

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow

Code

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3

int stack[MAX];
int top = -1;

void push(int);
void pop();
int isEmpty();
int isFull();
void display();

int main(){

    int choice,data;
    while(1){
        printf("press : 1 , to push element\n");
        printf("press : 2 , to pop element\n");
        printf("press : 3 ,  Display stack elements\n");
        printf("press : 4 , exit \n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
```

```
        switch(choice) {
            case 1:

                printf("Enter value to push : " );
                scanf("%d",&data);
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exited \n");
                exit(1);
            default :
                    printf("Invalid option or choice \n");
                    break;
        }

    }


    return 0 ;
}

void push(int data){
    if(isFull()) {
        printf("Stack overflow \n");
    }
    else{
        stack[++top] = data;
        printf("Element pushed to stack \n");
    }
```

```
}

void pop(){
    int temp ;
    if(isEmpty()){
         printf("Stack underflow \n");
    }
    else{
        temp = stack[top--];
        printf("Element Popped successfully \n");
    }
}
int isEmpty(){
    if(top == -1) return 1;
    else return 0;
}

int isFull(){

    if(top == MAX -1 ) return 1;
    else return 0;

}
void display(){
    if(isEmpty()){
         printf("Stack empty \n");
    }
    else{
        printf("Stack elements : ");
        for (int i =top; i > -1; i--){
            printf("%d ",stack[i]);
        }
          printf("\n");
    }
}
```

Output

```
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 1
Enter value to push : 2
Element pushed to stack
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 1
Enter value to push : 3
Element pushed to stack
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 3
Stack elements : 3 2
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 2
Element Popped successfully
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 3
Stack elements : 2
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 2
Element Popped successfully
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 3
Stack empty
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 2
Stack underflow
press : 1 , to push element
press : 2 , to pop element
press : 3 ,  Display stack elements
press : 4 , exit
Enter your choice : 4
Exited
```

Lab - 2

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

Code

```
#include <stdio.h>
#include <string.h>
int index1=0 , pos = 0 , top = -1 , length;
char symbol , temp , infix[20] , postfix[20] , stack[20];
void infixpostfix();
void push(char symbol);
char pop();
int pred(char symbol);

int main() {
    printf("Enter infix expression  : ");
    scanf("%s",&infix);
    printf("Infix exp : %s \n",infix);
    infixpostfix();
    printf("Postfix exp : %s \n",postfix);
    return 0;
}


void infixpostfix(){
    length = strlen(infix);
    push('#');
    while(index1 < length){
        symbol = infix[index1];
        switch(symbol){
            case '(': push(symbol);
                    break;
            case ')': temp = pop();
                    while(temp != '('){
```

```
                        postfix[pos++] = temp;
                        temp = pop();
                    }
                    break;

            case '+' :
                    case '-':
                    case '*':
                    case '/':
                    case '^':
                            while(pred(stack[top]) >=
pred(symbol)){
                                temp = pop();
                                postfix[pos++]=temp;
                            }
                            push(symbol);
                            break;
            default: postfix[pos++] = symbol;
        }
    index1++;
    }
    while(top > 0){
        temp = pop();
        postfix[pos++] = temp;
    }

}


void push(char symbol){
    stack[++top] = symbol;
}

char pop(){
    char symb;
    symb = stack[top--] ;
```

```
    return symb;
}

int pred(char symbol){
    int p ;
    switch(symbol){
        case '^': p=3;
        break;
        case '*':
        case '/': p=2;
            break;
        case '+':
        case '-': p=1;
        break;
        case '(': p = 0;
        break;
        case '#': p = -1;
        break;
    }
    return p;
}
```

Output

```
Enter infix expression  : ((a+b)^c-d/e*f)
Infix exp : ((a+b)^c-d/e*f)
Postfix exp : ab+c^de/f*-
```

Lab - 3a

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display
The program should print appropriate messages for queue empty and queue overflow conditions

Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int queue[MAX];
int front = -1, rear = -1;

void insert(int);
int delete();
void display();

int main() {

    int choice;
    printf("1 : insert \n");
    printf("2 : delete \n");
    printf("3 : display \n");
    printf("4 : exit \n");

    while (1) {
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int value;
                printf("Enter value: ");
                scanf("%d", &value);
```

```c
                insert(value);
                break;
            }
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid element\n");
                break;
        }
    }
    return 0;
}

void insert(int value) {
    if (front == -1 && rear == -1) {
        front = 0;
        rear = 0;
        queue[rear] = value;
    } else {
        if (rear >= MAX - 1) {
            printf("Overflow\n");
            return;
        } else {
            rear++;
            queue[rear] = value;
        }
    }
}
```

```c
int delete() {
    if (front == -1 && rear == -1) {
        printf("Empty, Underflow \n");
        return -1;
    } else {
        int deleted = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        printf("Deleted %d\n", deleted);
        return deleted;
    }
}

void display() {
    if (front == -1 && rear == -1) {
        printf("Empty, Underflow \n");
        return;
    } else {
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
```

Output

```
1 : insert
2 : delete
3 : display
4 : exit
Enter choice: 2
Empty, Underflow
Enter choice: 3
Empty, Underflow
Enter choice: 1
Enter value: 2
Enter choice: 1
Enter value: 3
Enter choice: 1
Enter value: 4
Enter choice: 1
Enter value: 5
Overflow
Enter choice: 3
2 3 4
Enter choice: 2
Deleted 2
Enter choice: 2
Deleted 3
Enter choice: 2
Deleted 4
Enter choice: 2
Empty, Underflow
Enter choice: 3
Empty, Underflow
Enter choice: 4
Exiting...
```

Lab - 3b

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display
The program should print appropriate messages for queue empty and queue overflow conditions

Code

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 3

int queue[MAX];
int front = -1, rear = -1;

void insert(int);
int delete();
void display();

int main() {
    int choice;
    printf("1 : insert \n");
    printf("2 : delete \n");
    printf("3 : display \n");
    printf("4 : exit \n");

    while (1) {
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int value;
                printf("Enter value: ");
                scanf("%d", &value);
                insert(value);
                break;
            }
            case 2:
                delete();
                break;
            case 3:
                display();
```

```
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid element\n");
                break;
        }
    }
    return 0;
}

void insert(int value) {
    if (front == -1 && rear == -1) { // Queue is empty
        front = 0;
        rear = 0;
        queue[rear] = value;
    } else {
        if ((rear + 1) % MAX == front) { // Check for overflow
            printf("Overflow\n");
            return;
        } else {
            rear = (rear + 1) % MAX;
            queue[rear] = value;
        }
    }
}

int delete() {
    if (front == -1 && rear == -1) { // Queue is empty
        printf("Empty, Underflow \n");
        return -1;
    } else {
        int deleted = queue[front];
        if (front == rear) { // Queue will be empty after this
```

```
delete
            front = rear = -1;
        } else {
            front = (front + 1) % MAX; // Move front forward
        }
        printf("Deleted %d\n", deleted);
        return deleted;
    }
}

void display() {
    if (front == -1 && rear == -1) {
        printf("Empty, Underflow \n");
        return;
    } else {
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) {
                break;
            }
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}
```

Output

```
1 : insert
2 : delete
3 : display
4 : exit
Enter choice: 2
Empty, Underflow
Enter choice: 3
Empty, Underflow
Enter choice: 1
Enter value: 2
Enter choice: 1
Enter value: 3
Enter choice: 1
Enter value: 4
Enter choice: 1
Enter value: 5
Overflow
Enter choice: 3
2 3 4
Enter choice: 2
Deleted 2
Enter choice: 2
Deleted 3
Enter choice: 2
Deleted 4
Enter choice: 2
Empty, Underflow
Enter choice: 3
Empty, Underflow
Enter choice: 4
Exiting...
```

Lab - 4 and 5

WAP to Implement Singly Linked List with following operations
a) Createalinkedlist.
b) Insertion of a node at first position, at any position and at end of list.
c) Deletion of first element, specified element and last element in the list.
d) Display the contents of the linked list.

Code

```
#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *next;
};

struct node *start = NULL;
struct node *create_ll(struct node*);
struct node *display(struct node*);
struct node *insert_beg(struct node*);
struct node *insert_end(struct node*);
struct node *insert_atPos(struct node*);
struct node *delete_beg(struct node*);
struct node *delete_end(struct node*);
struct node *delete_atPos(struct node*);

int main() {
    int choice;
    printf("\n1: Create LL\n2: Display\n3: Insert at Beginning\n4: Insert at End\n5: Insert at Position\n6: Delete from Beginning\n7: Delete from End\n8: Delete from Position\n9: Exit\n");
    int flag = 1;
    while (flag) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: start = create_ll(start); break;
            case 2: start = display(start); break;
            case 3: start = insert_beg(start); break;
            case 4: start = insert_end(start); break;
            case 5: start = insert_atPos(start); break;
            case 6: start = delete_beg(start); break;
            case 7: start = delete_end(start); break;
```

```c
            case 8: start = delete_atPos(start); break;
            case 9: flag = 0; break;
            default: printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}

struct node *create_ll(struct node *start) {
    struct node *new_node, *ptr;
    int num;
    printf("Enter num: ");
    scanf("%d", &num);
    while(num != -1) {
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num;
        if(start == NULL) {
            new_node->next = NULL;
            start = new_node;
        } else {
            ptr = start;
            while(ptr->next != NULL) ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        printf("Enter num: ");
        scanf("%d", &num);
    }
    return start;
}

struct node *display(struct node *start) {
    struct node *ptr;
    ptr = start;
    while (ptr != NULL) {
```

```c
        printf("\t %d", ptr->data);
        ptr = ptr->next;
    }
    return start;
}

struct node *insert_beg(struct node *start) {
    struct node *new_node;
    int num;
    printf("Enter num: ");
    scanf("%d", &num);
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
    return start;
}

struct node *insert_end(struct node *start) {
    struct node *new_node, *ptr;
    int num;
    printf("Enter num: ");
    scanf("%d", &num);
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    ptr = start;
    if(start == NULL) {
        start = new_node;
    } else {
        while(ptr->next != NULL) ptr = ptr->next;
        ptr->next = new_node;
    }
    return start;
}
```

```c
struct node *insert_atPos(struct node *start) {
    struct node *new_node, *ptr, *preptr;
    int num, indx = 0, pos;
    printf("Enter num: ");
    scanf("%d", &num);
    printf("Enter position: ");
    scanf("%d", &pos);

    if(pos < 0) {
        printf("Invalid position.\n");
        return start;
    }

    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;

    if(pos == 0) {
        new_node->next = start;
        start = new_node;
        return start;
    }

    while(ptr != NULL && indx < pos) {
        preptr = ptr;
        ptr = ptr->next;
        indx++;
    }

    if(ptr == NULL && indx < pos) {
        printf("Position is greater than the length of the
list.\n");
        free(new_node);
        return start;
```

```c
    }

    preptr->next = new_node;
    new_node->next = ptr;

    return start;
}

struct node *delete_beg(struct node *start) {
    struct node *ptr;
    ptr = start;
    start = start->next;
    free(ptr);
    return start;
}

struct node *delete_end(struct node *start) {
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr->next != NULL) {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = NULL;
    free(ptr);
    return start;
}

struct node *delete_atPos(struct node *start) {
    struct node *ptr, *preptr;
    int indx = 0, pos;
    printf("Enter position: ");
    scanf("%d", &pos);

    if(pos < 0 || start == NULL) {
```

```c
        printf("Invalid position or empty list.\n");
        return start;
    }

    ptr = start;
    preptr = NULL;

    if(pos == 0) {
        start = start->next;
        free(ptr);
        return start;
    }

    while(ptr != NULL && indx < pos) {
        preptr = ptr;
        ptr = ptr->next;
        indx++;
    }

    if(ptr == NULL) {
        printf("Position is greater than the length of the 
list.\n");
        return start;
    }

    preptr->next = ptr->next;
    free(ptr);

    return start;
}
```

Output

```
1: Create LL
2: Display
3: Insert at Beginning
4: Insert at End
5: Insert at Position
6: Delete from Beginning
7: Delete from End
8: Delete from Position
9: Exit

Enter your choice: 1
Enter num: 2
Enter num: 3
Enter num: 4
Enter num: 5
Enter num: 6
Enter num: -1

Enter your choice: 2
         2       3       4       5       6
Enter your choice: 7

Enter your choice: 2
         2       3       4       5
Enter your choice: 6

Enter your choice: 2
         3       4       5
Enter your choice: 3
Enter num: 2

Enter your choice: 2
         2       3       4       5
Enter your choice: 4
Enter num: 6

Enter your choice: 2
         2       3       4       5       6
Enter your choice: 5
Enter num: 0
Enter position: 2

Enter your choice: 2
         2       3       0       4       5       6
Enter your choice: 8
Enter position: 2

Enter your choice: 2
         2       3       4       5       6
Enter your choice: 9
```

Lab - 6a

WAP to Implement Single Link List with following operations: Sortthelinkedlist, Reversethelinkedlist, Concatenation of two linked lists.

Code

```
#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *next;
};

struct node *head1 = NULL;
struct node *head2 = NULL;
void create_ll();
struct node *display(struct node*);
struct node *sort(struct node*);
struct node *concat(struct node*, struct node*);
struct node *reverse(struct node*);

int main() {
    int choice;
    printf("\n1: Create LL\n2: Display\n3: Sort\n4:
Concat\n5: Reverse\n6: Exit\n");
    int flag = 1;
    while (flag) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                create_ll();  // Create both lists
                break;
            case 2: {
```

```
                int c;
                printf("Enter list 1 or 2: ");
                scanf("%d", &c);
                if (c == 1) {
                    head1 = display(head1);
                } else if (c == 2) {
                    head2 = display(head2);
                } else {
                    printf("Invalid list choice.\n");
                }
                break;
            }
            case 3: {
                int c;
                printf("Enter list 1 or 2: ");
                scanf("%d", &c);
                if (c == 1) {
                    head1 = sort(head1);
                } else if (c == 2) {
                    head2 = sort(head2);
                } else {
                    printf("Invalid list choice.\n");
                }
                break;
            }
            case 4: head1 = concat(head1, head2); break;
            case 5: head1 = reverse(head1); break;
            case 6: flag = 0; break;
            default: printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}

void create_ll() {
    struct node *new_node, *ptr;
```

```c
    int num;

    printf("Enter elements for list 1 (enter -1 to stop): ");
    while (1) {
        scanf("%d", &num);
        if (num == -1) break;
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num;
        new_node->next = NULL;
        if (head1 == NULL) {
            head1 = new_node;
        } else {
            ptr = head1;
            while (ptr->next != NULL) ptr = ptr->next;
            ptr->next = new_node;
        }
    }

    printf("Enter elements for list 2 (enter -1 to stop): ");
    while (1) {
        scanf("%d", &num);
        if (num == -1) break;
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num;
        new_node->next = NULL;
        if (head2 == NULL) {
            head2 = new_node;
        } else {
            ptr = head2;
            while (ptr->next != NULL) ptr = ptr->next;
            ptr->next = new_node;
        }
    }
}

struct node *display(struct node *head) {
```

```
    struct node *ptr = head;
    if (head == NULL) {
        printf("List is empty.\n");
        return head;
    }
    printf("List: ");
    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
    return head;
}

struct node *sort(struct node *head) {
    struct node *ptr, *cptr;
    int temp;
    for (ptr = head; ptr != NULL; ptr = ptr->next) {
        for (cptr = ptr->next; cptr != NULL; cptr =
cptr->next) {
            if (ptr->data > cptr->data) {
                temp = ptr->data;
                ptr->data = cptr->data;
                cptr->data = temp;
            }
        }
    }
    return head;
}
struct node *concat(struct node *head1, struct node *head2) {
    struct node *ptr = head1;
    if (head1 == NULL) return head2;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->next = head2;
```

```c
    return head1;
}
struct node *reverse(struct node *head) {
    struct node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

Output

```
1: Create LL
2: Display
3: Sort
4: Concat
5: Reverse
6: Exit

Enter your choice: 1
Enter elements for list 1 (enter -1 to stop): 1 4 3 5 -1
Enter elements for list 2 (enter -1 to stop): 6 2 7 -1

Enter your choice: 2
Enter list 1 or 2: 1
List: 1 4 3 5

Enter your choice: 2
Enter list 1 or 2: 2
List: 6 2 7

Enter your choice: 3
Enter list 1 or 2: 1

Enter your choice: 2
Enter list 1 or 2: 1
List: 1 3 4 5

Enter your choice: 3
Enter list 1 or 2: 2

Enter your choice: 2
Enter list 1 or 2: 2
List: 2 6 7

Enter your choice: 4

Enter your choice: 2
Enter list 1 or 2: 1
List: 1 3 4 5 2 6 7

Enter your choice: 5

Enter your choice: 2
Enter list 1 or 2: 1
List: 7 6 2 5 4 3 1

Enter your choice: 6
```

Lab - 6b

WAP to Implement Single Link List to simulate Stack & Queue Operations.

Stack Code

```
#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *next;
};

struct node *top = NULL;

struct node* push(struct node*, int);
struct node* delete(struct node*);
void display(struct node*);

int main() {
    int choice;
    printf("\n1: Insert\n2: Delete\n3: Display\n4: Exit\n");
    int flag = 1;
    while (flag) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: {
                int data;
                printf("Enter data: ");
                scanf("%d", &data);
                top = push(top, data);
                break;
            }
            case 2:
                top = delete(top);
                break;
            case 3:
                display(top);
```

```
                break;
            case 4:
                flag = 0;
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}

struct node* push(struct node* top, int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));
    if (!new_node) {
        printf("Memory allocation failed!\n");
        return top;
    }
    new_node->data = data;
    new_node->next = top;
    top = new_node;
    return top;
}

struct node* delete(struct node* top) {
    if (top == NULL) {
        printf("Underflow: The stack is empty.\n");
        return top;
    }
    struct node* temp = top;
    top = top->next;
    free(temp);
    return top;
}
void display(struct node* top) {
    if (top == NULL) {
```

```c
        printf("The stack is empty.\n");
        return;
    }
    struct node* ptr = top;
    printf("Stack elements: ");
    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

Output

```
1: Insert
2: Delete
3: Display
4: Exit

Enter your choice: 3
The stack is empty.

Enter your choice: 2
Underflow: The stack is empty.

Enter your choice: 1
Enter data: 2

Enter your choice: 1
Enter data: 3

Enter your choice: 1
Enter data: 4

Enter your choice: 3
Stack elements: 4 3 2

Enter your choice: 2

Enter your choice: 2

Enter your choice: 2

Enter your choice: 2
Underflow: The stack is empty.

Enter your choice: 3
The stack is empty.

Enter your choice: 4
```

Queue Code

```c
#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *next;
};

struct node *front = NULL, *rear = NULL;

void enqueue(int data);
void dequeue();
void display();

int main() {
    int choice, data, flag = 1;

    printf("\nQueue Operations using Singly Linked List");
    printf("\n1: Enqueue\n2: Dequeue\n3: Display\n4: Exit\n");

    while (flag) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &data);
                enqueue(data);
                break;
            case 2:
                dequeue();
                break;
            case 3:
```

```
                display();
                break;
            case 4:
                flag = 0;
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}

void enqueue(int data) {
    struct node *new_node = (struct node*)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->next = NULL;

    if (rear == NULL) {
        front = rear = new_node;
    } else {
        rear->next = new_node;
        rear = new_node;
    }
    printf("Enqueued: %d\n", data);
}

void dequeue() {
    if (front == NULL) {
        printf("Underflow: The queue is empty.\n");
        return;
    }
    struct node *temp = front;
    front = front->next;
```

```c
    if (front == NULL) {
        rear = NULL;
    }

    printf("Dequeued: %d\n", temp->data);
    free(temp);
}

void display() {
    if (front == NULL) {
        printf("The queue is empty.\n");
        return;
    }

    struct node *ptr = front;
    printf("Queue elements: ");
    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

Output

```
Queue Operations using Singly Linked List
1: Enqueue
2: Dequeue
3: Display
4: Exit

Enter your choice: 3
The queue is empty.

Enter your choice: 2
Underflow: The queue is empty.

Enter your choice: 1
Enter data: 1
Enqueued: 1

Enter your choice: 1
Enter data: 2
Enqueued: 2

Enter your choice: 1
Enter data: 3
Enqueued: 3

Enter your choice: 1
Enter data: 4
Enqueued: 4

Enter your choice: 3
Queue elements: 1 2 3 4

Enter your choice: 2
Dequeued: 1

Enter your choice: 2
Dequeued: 2

Enter your choice: 2
Dequeued: 3

Enter your choice: 2
Dequeued: 4

Enter your choice: 2
Underflow: The queue is empty.

Enter your choice: 3
The queue is empty.

Enter your choice: 4
```

Lab - 7

WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

Code

```
#include <stdio.h>
#include <malloc.h>

struct node
{
    int data;
    struct node *prev;
    struct node *next;
};

struct node *start = NULL;
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_before(struct node *);
struct node *delete_node(struct node *);

int main()
{
    int choice, flag = 1;
    printf("1.Create DLL \n2.Insert before \n3. Delete Node\n4.
Display\n5.Exit\n");

    while (flag)
    {
        printf("Enter choice: ");
        scanf("%d", &choice);
```

```
        switch (choice)
        {
        case 1:
            start = create_ll(start);
            break;
        case 2:
            start = insert_before(start);
            break;
        case 3:
        start=delete_node(start);
        break;
        case 4:
            start = display(start);
            break;
        case 5:
            flag = 0;
            break;
        default:
            printf("Invalid choice! Please enter a valid
option.\n");
            break;
        }
    }
}

struct node *create_ll(struct node *start)
{
    struct node *new_node, *ptr;
    int number;
    printf("enter -1 to end \n");
    printf("enter the data:");
    scanf("%d", &number);
    while (number != -1)
    {
        new_node = (struct node *)malloc(sizeof(struct node));
```

```
        new_node->data = number;
        if (start == NULL)
        {
            new_node->next = NULL;
            new_node->prev = NULL;
            start = new_node;
        }
        else
        {
            ptr = start;
            while (ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = new_node;
            new_node->prev = ptr;
            new_node->next = NULL;
        }
        printf("enter the data:");
        scanf("%d", &number);
    }
    return start;
}


struct node* insert_before(struct node *start) {
    int num, value;
    printf("Enter Number to insert Before\n");
    scanf("%d", &num);
    printf("Enter value to insert\n");
    scanf("%d", &value);

    struct node *ptr = start;
    struct node *new_node = (struct node *)malloc(sizeof(struct
node));
```

```
    new_node->data = value;

    if (start == NULL) {
        new_node->next = NULL;
        new_node->prev = NULL;
        return new_node;
    }
    while (ptr != NULL && ptr->data != num) {
        ptr = ptr->next;
    }
    if (ptr == NULL) {
        printf("Node with value %d not found in the list.\n", num);
        free(new_node);
        return start;
    }
    if (ptr == start) {
        new_node->next = start;
        new_node->prev = NULL;
        start->prev = new_node;
        return new_node;
    }
    new_node->next = ptr;
    new_node->prev = ptr->prev;
    ptr->prev->next = new_node;
    ptr->prev = new_node;

    return start;
}
struct node *delete_node(struct node *start) {
    struct node *ptr, *temp;
    int val;
    if (start == NULL) {
        printf("List is empty.\n");
        return start;
    }
```

```
    printf("Enter the data to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    while (ptr != NULL && ptr->data != val) {
        ptr = ptr->next;
    }
    if (ptr == NULL) {
        printf("Value not found in the list.\n");
        return start;
    }
    if (ptr == start) {
        start = start->next;
        if (start != NULL) start->prev = NULL;
        free(ptr);
    } else {
        ptr->prev->next = ptr->next;
        if (ptr->next != NULL) ptr->next->prev = ptr->prev;
        free(ptr);
    }
    return start;
}
struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        printf(" %d", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
    return start;
}
```

Output

```
1.Create DLL
2.Insert before
3. Delete Node
4. Display
5.Exit
Enter choice: 1
enter -1 to end
enter the data:1
enter the data:2
enter the data:3
enter the data:4
enter the data:-1
Enter choice: 4
 1 2 3 4
Enter choice: 3
Enter the data to be deleted: 2
Enter choice: 4
 1 3 4
Enter choice: 3
Enter the data to be deleted: 4
Enter choice: 4
 1 3
Enter choice: 3
Enter the data to be deleted: 1
Enter choice: 4
 3
Enter choice: 2
Enter Number to insert Before
1
Enter value to insert
2
Node with value 1 not found in the list.
Enter choice: 2
Enter Number to insert Before
3
Enter value to insert
 1 3
Enter choice: 3
Enter the data to be deleted: 1
Enter choice: 4
 3
Enter choice: 2
Enter Number to insert Before
1
Enter value to insert
2
Node with value 1 not found in the list.
Enter choice: 2
Enter Number to insert Before
3
Enter value to insert
1
Enter choice: 4
 1 3
Enter choice: 5
```

Lab - 8

Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order
c) To display the elements in the tree.

Code

```
#include <stdio.h>
#include <malloc.h>

typedef struct BST {
    int data;
    struct BST *left;
    struct BST *right;
} node;

node *create() {
    node *temp;
    printf("Enter data: ");
    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}

void insert(node *root, node *temp) {
    if (temp->data < root->data) {
        if (root->left != NULL)
            insert(root->left, temp);
        else
            root->left = temp;
    } else if (temp->data > root->data) {
        if (root->right != NULL)
```

```
            insert(root->right, temp);
        else
            root->right = temp;
    }
}

void preorder(node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void postorder(node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    char ch;
    int n = 1;
    node *root = NULL, *temp;
    do {
```

```
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);
        printf("\nEnter 0 to exit ");
        scanf("%d",&n);
    } while (n!=0);

    printf("\nPreorder Traversal: ");
    preorder(root);
    printf("\nInorder Traversal: ");
    inorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);
    return 0;
}
```

Output

```
Enter data: 7

Enter 0 to exit 1
Enter data: 3

Enter 0 to exit 1
Enter data: 9

Enter 0 to exit 1
Enter data: 1

Enter 0 to exit 1
Enter data: 4

Enter 0 to exit 1
Enter data: 8

Enter 0 to exit 1
Enter data: 10

Enter 0 to exit 0

Preorder Traversal: 7 3 1 4 9 8 10
Inorder Traversal: 1 3 4 7 8 9 10
Postorder Traversal: 1 4 3 8 10 9 7
```

Lab - 9a

Write a program to traverse a graph using BFS method.

Code

```
#include <stdio.h>

void bfs(int adj[10][10], int n, int source){
    int que[10];
    int front=0,rear=-1;
    int visited[10]={0};
    int node;
    printf("The nodes visited from %d: ", source);
    que[++rear]=source;
    visited[source]=1;
    printf("%d",source);
    while(front<=rear){
        int u= que[front++];
        for(int v=0; v<n; v++){
            if(adj[u][v]==1){
                if(visited[v]==0){
                    printf("%d",v);
                    visited[v]=1;
                    que[++rear]=v;
                }
            }
        }
    }
    printf("\n");
}

int main() {
    int n;
    int adj[10][10];
    int source;
    printf("enter number of nodes \n");
```

```
    scanf("%d",&n);
    printf("Enter Adjacency Matrix \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d",&adj[i][j]);
        }
    }
    for(source=0; source<n; source++){
        bfs(adj,n,source);
    }

    return 0;
}
```

Output

```
enter number of nodes
5
Enter Adjacency Matrix
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0\
The nodes visited from 0: 01324
The nodes visited from 1: 10243
The nodes visited from 2: 21304
The nodes visited from 3: 30241
The nodes visited from 4: 41302
```

Lab - 9b

Write a program to check whether given graph is connected or not using DFS method.

Code

```c
#include <stdio.h>

#include <stdbool.h>


#define MAX 100


int adjMatrix[MAX][MAX];

bool visited[MAX];

int stack[MAX];

int top = -1;


void push(int vertex) {

    if (top == MAX - 1) {

        printf("Stack Overflow\n");

        return;

    }

    stack[++top] = vertex;

}


int pop() {

    if (top == -1) {

        printf("Stack Underflow\n");

        return -1;

    }
```

```
    return stack[top--];
}

void dfsUsingStack(int startVertex, int numVertices) {
    push(startVertex);
    visited[startVertex] = true;
    while (top != -1) {
        int currentVertex = pop();

        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                push(i);
                visited[i] = true;
            }
        }
    }
}
bool isConnected(int numVertices) {
    for (int i = 0; i < numVertices; i++) {
        visited[i] = false;
    }
    dfsUsingStack(0, numVertices);
    for (int i = 0; i < numVertices; i++) {
        if (!visited[i]) {
```

```
            return false;
        }
    }
    return true;
}
int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
      }
    }
    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < numEdges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }
    if (isConnected(numVertices)) {
```

```c
        printf("The graph is connected.\n");

    } else {

        printf("The graph is not connected.\n");

    }

    return 0;

}
```

Output

Case - 1

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (start_vertex end_vertex):
0 1
0 2
1 2
1 4
2 3
3 4
The graph is connected.
```

Case -2

```
Enter the number of vertices: 4
Enter the number of edges: 2
Enter the edges (start_vertex end_vertex):
0 1
2 3
The graph is not connected.
```

Lab - 10

Given a File of N employee records with a set K of Keys(4- digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.
Let the keys in K and addresses in L are integers.

Code

```
#include <stdio.h>

#define MAX 100


struct Employee {

    int k;

    char n[50];

};


struct Employee ht[MAX];

int ts;


void init() {

    for (int i = 0; i < MAX; i++) ht[i].k = -1;

}


int hash(int k) {

    return k % ts;

}


void insert(int k, char n[]) {

    int idx = hash(k);

    while (ht[idx].k != -1) {
```

```
        idx = (idx + 1) % ts;
    }
    ht[idx].k = k;
    for (int i = 0; n[i] != '\0' && i < 49; i++) {
        ht[idx].n[i] = n[i];
    }
    ht[idx].n[49] = '\0';
}

void display() {
    for (int i = 0; i < ts; i++) {
        if (ht[i].k != -1)
            printf("Idx %d: Key = %d, Name = %s\n", i, ht[i].k,
ht[i].n);
        else
            printf("Idx %d: Empty\n", i);
    }
}

int main() {
    int n;
    printf("Enter table size (max size %d): ", MAX);
    scanf("%d", &ts);
```

```c
    if (ts > MAX) ts = MAX;

    init();

    printf("Enter number of employees: ");
    scanf("%d", &n);
    getchar();

    for (int i = 0; i < n; i++) {
        int k;
        char name[50];
        printf("Enter key and name for employee %d: ", i + 1);
        scanf("%d", &k);
        getchar();
        gets(name);
        insert(k, name);
    }

    display();

    return 0;
}
```

Output

```
Enter table size (max size 100): 7
Enter number of employees: 5
Enter key and name for employee 1: 1256 ram
Enter key and name for employee 2: 1452 sham
Enter key and name for employee 3: 9845 raju
Enter key and name for employee 4: 6374 adi
Enter key and name for employee 5: 4778 lal
Idx 0: Key = 4778, Name = lal
Idx 1: Empty
Idx 2: Empty
Idx 3: Key = 1256, Name = ram
Idx 4: Key = 1452, Name = sham
Idx 5: Key = 9845, Name = raju
Idx 6: Key = 6374, Name = adi
```

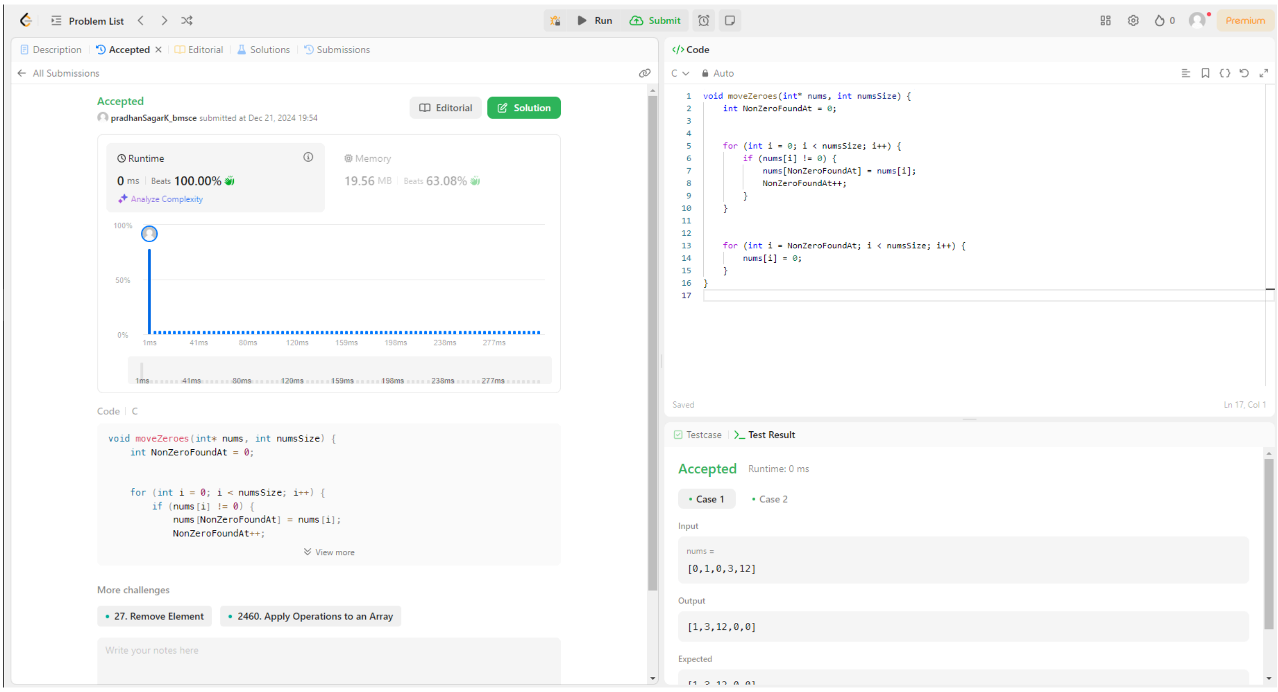
LeetCode Problems

LeetCode - 283 Move Zeroes

```
void moveZeroes(int* nums, int numsSize) {
    int NonZeroFoundAt = 0;



    for (int i = 0; i < numsSize; i++) {
        if (nums[i] != 0) {
            nums[NonZeroFoundAt] = nums[i];
            NonZeroFoundAt++;
        }
    }
```

```
    for (int i = NonZeroFoundAt; i < numsSize; i++) {
        nums[i] = 0;
    }
}
```

Output



Hacker Rank - Game of two stacks

```
int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
int s1 = 0, s2 = 0, maxCount = 0, i = 0, j = 0;

    while (i < a_count && s1 + a[i] <= maxSum) {
        s1 += a[i];
        i++;
```

```
    }
    maxCount = i;

    while (j < b_count && i >= 0) {
        s2 += b[j];
        j++;
        while (s1 + s2 > maxSum && i > 0) {
            s1 -= a[--i];
        }
        if (s1 + s2 <= maxSum) {
            if (i + j > maxCount) {
                maxCount = i + j;
            }
        }
    }
    return maxCount;
}
```

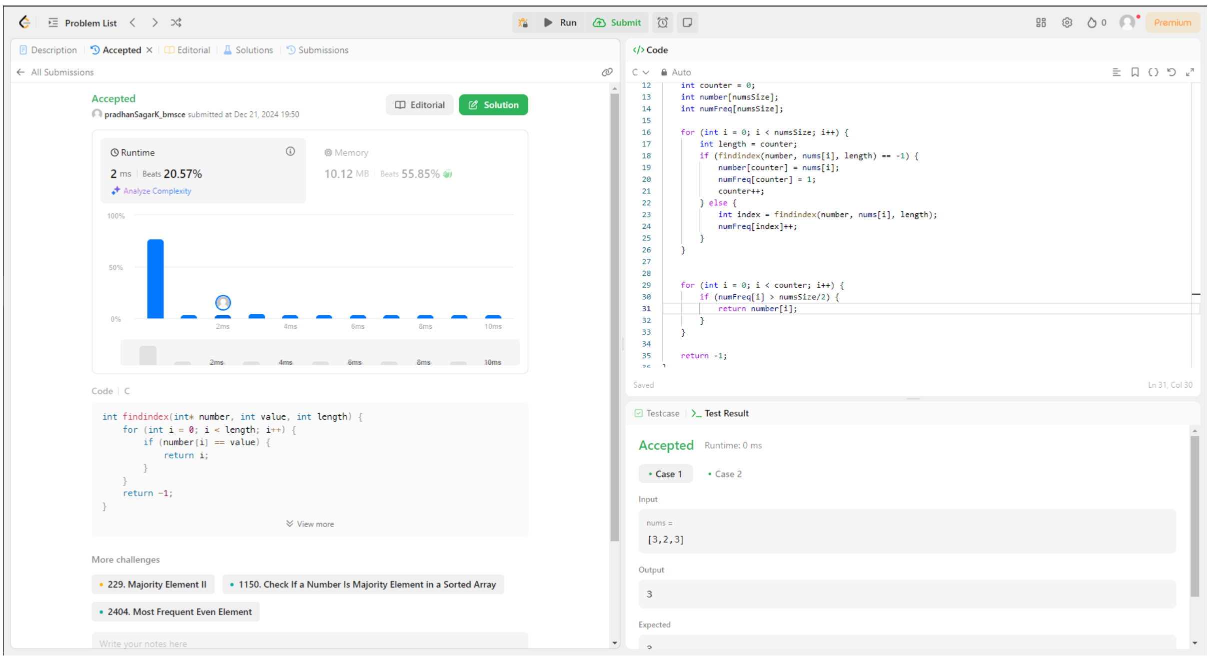
Output

LeetCode - 169 Majority Element

```
int findindex(int* number, int value, int length) {
    for (int i = 0; i < length; i++) {
        if (number[i] == value) {
            return i;
        }
    }
    return -1;
}

int majorityElement(int* nums, int numsSize) {

    int counter = 0;
    int number[numsSize];
    int numFreq[numsSize];

    for (int i = 0; i < numsSize; i++) {
        int length = counter;
        if (findindex(number, nums[i], length) == -1) {
            number[counter] = nums[i];
            numFreq[counter] = 1;
            counter++;
        } else {
            int index = findindex(number, nums[i], length);
```

```
            numFreq[index]++;
        }
    }


    for (int i = 0; i < counter; i++) {
        if (numFreq[i] > numsSize/2) {
            return number[i];
        }
    }

    return -1;
}
```

Output

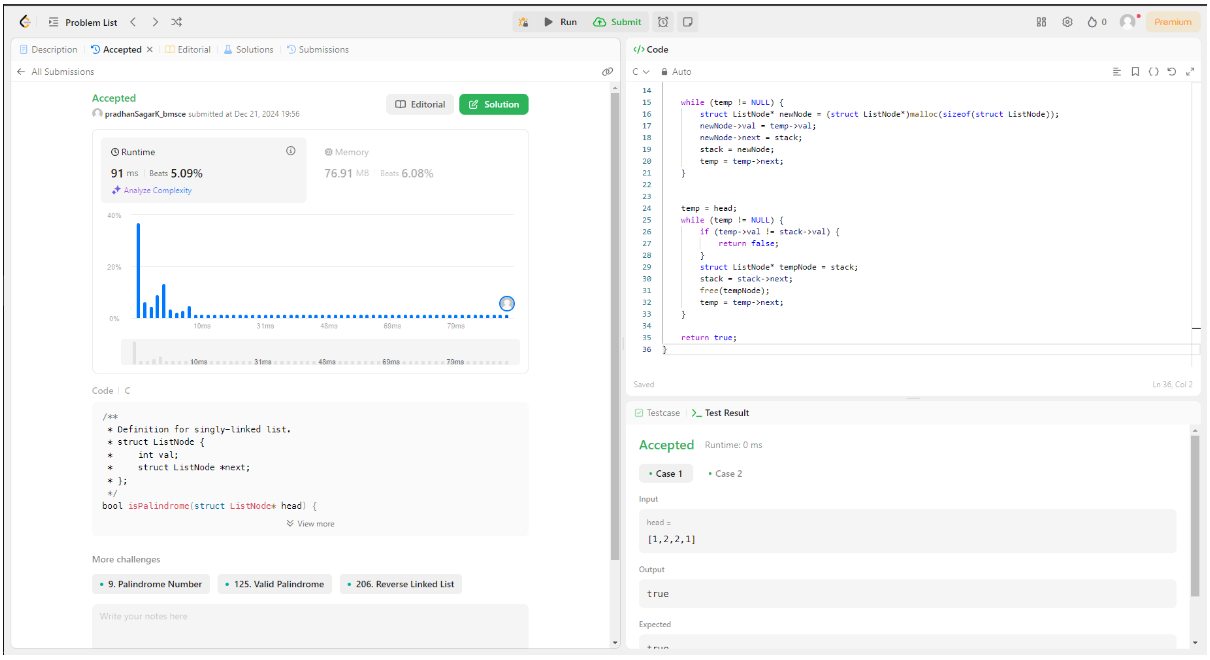


LeetCode - 234 Palindrome Linked List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool isPalindrome(struct ListNode* head) {
   if (head == NULL || head->next == NULL) return true;

   struct ListNode* temp = head;
   struct ListNode* stack = NULL;



   while (temp != NULL) {
       struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct
ListNode));
       newNode->val = temp->val;
       newNode->next = stack;
       stack = newNode;
       temp = temp->next;
   }
   temp = head;
   while (temp != NULL) {
```

```
        if (temp->val != stack->val) {
            return false;
        }
        struct ListNode* tempNode = stack;
        stack = stack->next;
        free(tempNode);
        temp = temp->next;
    }
    return true;
}
```

Output



LeetCode - 112 Path Sum

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool hasPathSum(struct TreeNode* root, int targetSum) {
    if (root == NULL)
        return false;
    if (root->val == targetSum && root->left == NULL && root->right == NULL)
        return true;
    return hasPathSum(root->left, targetSum - root->val) ||
           hasPathSum(root->right, targetSum - root->val);
}
```

Output